Express.js API Practice - CCIT 4th Semester

Date: Thursday, 5th February 2026
From: Muhamad Hudya Ramadhana S.Tr.Kom
Phone: +62 822 1330 8462

Getting Started with Express.js

Dear my students, in this session you will learn how to build a REST API using Express.js. We will start by cloning a boilerplate repository and understanding its structure. Then, you will implement CRUD operations using an in-memory array (before we move to database in the next sessions).

Step 1: Clone the Boilerplate

Repository URL

`https://github.com/perogeremmer/boilerplate-express-ccit`

Clone Command

Open your terminal and run:

```
git clone https://github.com/perogeremmer/boilerplate-express-ccit.git
cd boilerplate-express-ccit
```

Install Dependencies

```
npm install
```

Project Structure Overview

After cloning, you will see this structure:

```
src/
├── controllers/
│   └── main-controller.js    # Route handlers/logic
├── routes/
│   └── index.js              # API routes definitions
├── routes.test.js            # Test file
└── index.js                  # Application entry point
```

Step 2: Understanding the Structure

File: src/index.js

This is the entry point of the application. It:

- Creates the Express app
- Sets up middleware
- Imports routes
- Starts the server

File: src/routes/index.js

This file defines all API endpoints (URLs) and maps them to controller functions.

File: src/controllers/main-controller.js

This file contains the actual logic that runs when someone hits an endpoint.

Step 3: Run the Application

Development Mode (with auto-restart)

```
npm run dev
```

Check if Running

Open browser or use curl:

```
curl http://localhost:3000/
curl http://localhost:3000/health
```

Expected response for `/`:

```
{
  "message": "Welcome to Express API"
}
```

Expected response for `/health`:

```
{
  "status": "OK",
  "timestamp": "2026-02-05T00:00:00.000Z"
}
```

Step 4: Your First Task - Create Simple Routes

Task 4.1: Add a About Route

Create a new route `GET /about` that returns information about the API.

In `src/routes/index.js`, add:

```
import { about } from '../controllers/main-controller.js';

// Add this line with other routes
router.get('/about', about);
```

In `src/controllers/main-controller.js`, add:

```
export const about = (req, res) => {
  res.json({
    name: "CCIT Student API",
    version: "1.0.0",
    author: "Your Name",
    description: "A simple API for learning Express.js"
  });
};
```

Test it:

```
curl http://localhost:3000/about
```

Expected response:

```
{
  "name": "CCIT Student API",
  "version": "1.0.0",
  "author": "Your Name",
  "description": "A simple API for learning Express.js"
}
```

Task 4.2: Add a Route with URL Parameter

Create a route `GET /greet/:name` that greets a person by name.

In `src/routes/index.js`, add:

```
import { greet } from '../controllers/main-controller.js';

router.get('/greet/:name', greet);
```

In `src/controllers/main-controller.js`, add:

```
export const greet = (req, res) => {
  const name = req.params.name;
  res.json({
    message: `Hello, ${name}! Welcome to CCIT API.`
  });
};
```

Test it:

```
curl http://localhost:3000/greet/Budi
```

Expected response:

```
{
  "message": "Hello, Budi! Welcome to CCIT API."
}
```

Step 5: CRUD with Array (In-Memory Storage)

Now we will build a complete CRUD API for managing "Products" using an array as temporary storage.

Understanding CRUD

- Create: Add new data (POST)
- Read: Get data (GET)
- Update: Modify existing data (PUT/PATCH)
- Delete: Remove data (DELETE)

Step 5.1: Setup the Array

In `src/controllers/main-controller.js`, add at the top:

```
// In-memory storage for products
let products = [
  { id: 1, name: "Laptop", price: 10000000, stock: 10 },
  { id: 2, name: "Mouse", price: 150000, stock: 50 },
  { id: 3, name: "Keyboard", price: 300000, stock: 30 }
];

// Helper function to generate new ID
const generateId = () => {
  return products.length > 0 ? Math.max(...products.map(p => p.id)) + 1 : 1;
};
```

Step 5.2: GET All Products (READ)

In `src/controllers/main-controller.js`:

```
export const getAllProducts = (req, res) => {
  res.json({
```

```
    success: true,
    count: products.length,
    data: products
  });
};
```

In `src/routes/index.js`:

```
import { getAllProducts } from '../controllers/main-controller.js';

router.get('/products', getAllProducts);
```

Test:

```
curl http://localhost:3000/products
```

Expected response:

```
{
  "success": true,
  "count": 3,
  "data": [
    { "id": 1, "name": "Laptop", "price": 10000000, "stock": 10 },
    { "id": 2, "name": "Mouse", "price": 150000, "stock": 50 },
    { "id": 3, "name": "Keyboard", "price": 300000, "stock": 30 }
  ]
}
```

Step 5.3: GET Single Product by ID (READ)

In `src/controllers/main-controller.js`:

```
export const getProductById = (req, res) => {
  const id = parseInt(req.params.id);
  const product = products.find(p => p.id === id);

  if (!product) {
    return res.status(404).json({
      success: false,
      message: `Product with id ${id} not found`
    });
  }

  res.json({
    success: true,
    data: product
  });
};
```

In `src/routes/index.js`:

```
import { getProductById } from '../controllers/main-controller.js';

router.get('/products/:id', getProductById);
```

Test:

```
curl http://localhost:3000/products/1
```

Expected response:

```
{
  "success": true,
  "data": { "id": 1, "name": "Laptop", "price": 10000000, "stock": 10 }
}
```

Test not found:

```
curl http://localhost:3000/products/999
```

Expected response (404):

```
{
  "success": false,
  "message": "Product with id 999 not found"
}
```

Step 5.4: POST Create New Product (CREATE)

In `src/controllers/main-controller.js`:

```
export const createProduct = (req, res) => {
  const { name, price, stock } = req.body;

  // Validation
  if (!name || !price || stock === undefined) {
    return res.status(400).json({
      success: false,
      message: "Please provide name, price, and stock"
    });
  }

  const newProduct = {
    id: generateId(),
    name: name,
    price: parseInt(price),
    stock: parseInt(stock)
  };

  products.push(newProduct);

  res.status(201).json({
    success: true,
    message: "Product created successfully",
    data: newProduct
  });
};
```

In `src/routes/index.js`:

```
import { createProduct } from '../controllers/main-controller.js';

router.post('/products', createProduct);
```

Note: The boilerplate should already have `express.json()` middleware in `src/index.js`:

```
app.use(express.json());
```

Test:

```
curl -X POST http://localhost:3000/products \
  -H "Content-Type: application/json" \
  -d '{"name":"Monitor","price":2000000,"stock":20}'
```

Expected response (201):

```
{
  "success": true,
  "message": "Product created successfully",
  "data": { "id": 4, "name": "Monitor", "price": 2000000, "stock": 20 }
}
```

Step 5.5: PUT Update Product (UPDATE)

In `src/controllers/main-controller.js`:

```
export const updateProduct = (req, res) => {
  const id = parseInt(req.params.id);
  const { name, price, stock } = req.body;

  const productIndex = products.findIndex(p => p.id === id);

  if (productIndex === -1) {
    return res.status(404).json({
      success: false,
      message: `Product with id ${id} not found`
    });
  }

  // Update only provided fields
  if (name) products[productIndex].name = name;
  if (price) products[productIndex].price = parseInt(price);
  if (stock !== undefined) products[productIndex].stock = parseInt(stock);

  res.json({
    success: true,
    message: "Product updated successfully",
    data: products[productIndex]
  });
};
```

In `src/routes/index.js`:

```
import { updateProduct } from '../controllers/main-controller.js';

router.put('/products/:id', updateProduct);
```

Test:

```
curl -X PUT http://localhost:3000/products/1 \
  -H "Content-Type: application/json" \
  -d '{"price":12000000,"stock":5}'
```

Expected response:

```
{
  "success": true,
  "message": "Product updated successfully",
  "data": { "id": 1, "name": "Laptop", "price": 12000000, "stock": 5 }
}
```

Step 5.6: DELETE Remove Product (DELETE)

In `src/controllers/main-controller.js`:

```
export const deleteProduct = (req, res) => {
  const id = parseInt(req.params.id);
  const productIndex = products.findIndex(p => p.id === id);

  if (productIndex === -1) {
    return res.status(404).json({
      success: false,
      message: `Product with id ${id} not found`
    });
  }

  const deletedProduct = products.splice(productIndex, 1)[0];

  res.json({
    success: true,
    message: "Product deleted successfully",
    data: deletedProduct
  });
};
```

In `src/routes/index.js`:

```
import { deleteProduct } from '../controllers/main-controller.js';

router.delete('/products/:id', deleteProduct);
```

Test:

```
curl -X DELETE http://localhost:3000/products/2
```

Expected response:

```
{
  "success": true,
  "message": "Product deleted successfully",
  "data": { "id": 2, "name": "Mouse", "price": 150000, "stock": 50 }
}
```

Step 6: Testing All Endpoints

Here is a complete test flow:

1. Get All Products

```
curl http://localhost:3000/products
```

2. Get Single Product

```
curl http://localhost:3000/products/1
```

3. Create New Product

```
curl -X POST http://localhost:3000/products \
  -H "Content-Type: application/json" \
  -d '{"name":"Webcam","price":500000,"stock":15}'
```

4. Update Product

```
curl -X PUT http://localhost:3000/products/1 \
  -H "Content-Type: application/json" \
  -d '{"stock":8}'
```

5. Delete Product

```
curl -X DELETE http://localhost:3000/products/3
```

6. Verify Delete

```
curl http://localhost:3000/products
```

Your Main Task: Student Management API

Requirements

Create a complete CRUD API for managing students with the following specifications:

Data Structure

Each student should have:

- `id` (number, auto-generated)
- `name` (string, required)
- `email` (string, required)
- `major` (string, required)
- `semester` (number, required)
- `gpa` (number, optional, default: 0.0)

Endpoints to Implement

1. `GET /students` - Get all students
   - Response: `{ success: true, count: X, data: [...] }`
2. `GET /students/:id` - Get single student by ID
   - Response: `{ success: true, data: {...} }`
   - Error (404): `{ success: false, message: "Student not found" }`
3. `POST /students` - Create new student
   - Required fields: name, email, major, semester
   - Response (201): `{ success: true, message: "...", data: {...} }`
   - Error (400): `{ success: false, message: "Please provide..." }`
4. `PUT /students/:id` - Update student
   - Can update any field
   - Response: `{ success: true, message: "...", data: {...} }`
5. `DELETE /students/:id` - Delete student
   - Response: `{ success: true, message: "...", data: {...} }`

Sample Data (Initial)

```
let students = [
  { id: 1, name: "Budi Santoso", email: "budi@ccit.edu", major: "Software
Engineering", semester: 4, gpa: 3.5 },
  { id: 2, name: "Ani Wijaya", email: "ani@ccit.edu", major: "Data Science",
semester: 4, gpa: 3.8 },
  { id: 3, name: "Citra Dewi", email: "citra@ccit.edu", major: "Software
Engineering", semester: 2, gpa: 3.2 }
];
```

Additional Challenge (Bonus)
Add these filter endpoints:

1. `GET /students/major/:major` - Get students by major
   - Example: `/students/major/Software%20Engineering`
2. `GET /students/semester/:semester` - Get students by semester
   - Example: `/students/semester/4`

Submission Requirements

GitHub Repository

1. Fork or create a new repository from the boilerplate
2. Implement all required features
3. Push your code to GitHub

README.md Documentation

Your README must include:
1. Project title and description
2. How to install and run
3. List of all available endpoints
4. Example requests and responses for each endpoint

Example format:

```
## API Endpoints

### Get All Students
- URL: GET /students
- Response:
```

json { “success”: true, “count”: 3, “data”: [...] }

Scoring Criteria

- **Code Functionality (40%)**: All endpoints work correctly
- **Error Handling (20%)**: Proper validation and error responses
- **Code Organization (15%)**: Clean structure and naming
- **Documentation (15%)**: Clear README with examples
- **Git Commits (10%)**: Regular commits with meaningful messages

Live Defense Preparation

Be ready to:
1. Explain how array methods work (`find`, `findIndex`, `push`, `splice`)
2. Debug when routes return wrong responses
3. Fix validation logic errors
4. Explain HTTP status codes (200, 201, 400, 404)

Important Notes

1. **DO NOT** use database yet - use array only
2. **DO NOT** use AI to generate the entire code - understand each line
3. **TEST** every endpoint using curl or Postman before submitting

4. **COMMIT** your progress regularly to GitHub
5. **ASK** questions if you're stuck - don't wait until the last minute

Remember: **Understanding is more important than completing.**

Good luck and happy coding! 🚀